# **Setting up Kubernetes Environment**

## **Play with k8s:**

### 1. Create four instances. One for master and three for worker nodes.

Required "Docker" or "Github" account.

<https://labs.play-with-k8s.com/>

### 2. Initialize Master node

# kubeadm init --apiserver-advertise-address $(hostname -i)

# kubectl apply -n kube-system -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 |tr -d '\n')"

### 3. Join worker node to the cluster

# kubeadm join --token [...] --discovery-token-ca-cert-hash [...]

### 4. Testing

# kubectl get no

# kubectl apply -f https://raw.githubusercontent.com/kubernetes/website/master/content/en/examples/application/nginx-app.yaml

# kubectl get po

## **Minikube:**

### 1. Download & Install "VirtualBox"

Source: <https://www.virtualbox.org/>

### 2. Download “minikube-windows-amd64”

C:\>mkdir Kubernetes

source: <https://github.com/kubernetes/minikube/releases>

### 3. Download “kubectl”

C:\Kubernetes> curl -LO https://storage.googleapis.com/kubernetes-release/release/v1.11.0/bin/windows/amd64/kubectl.exe

### 4. Start Minikube

C:\Kubernetes>minikube.exe start

### 5. Testing

C:\Kubernetes>minikube status

C:\Kubernetes>minikube version

C:\Kubernetes>kubectl get nodes

C:\Kubernetes>kubectl run kubernetes-bootcamp --image=gcr.io/google-samples/kubernetes-bootcamp:v1 --port=8080

C:\Kubernetes>kubectl get po

C:\Kubernetes>minikube stop

## **Kubeadm:**

### 1. Create 4 VMs - master (1) and worker (3) nodes

Node Requirements (varies depending on usage)

-------------------------------------

Master: 2 vCPUs - 6GB Ram

Worker: 1 vCPUs - 3GB RAM

OS: CentOS/RHEL 7

### 2. PRE-Reqs: Disable - Firewall | Swap | SELinux

Note: Execute on all nodes (master & worker)

* Disable Firewall (OR - ensure ports [6443 10250] are open )

# systemctl stop firewalld

# systemctl disable firewalld

* Disable Swap

# swapoff -a

# sed -i.bak -r 's/(.+ swap .+)/#\1/' /etc/fstab

* Disable SELinux

# setenforce 0

# sed -i 's/enforcing/disabled/g' /etc/selinux/config

### 3. Download & Install - Docker | Kubelet | Kubeadm | Kubectl

- Note: Execute on all nodes (master & worker)

- Kubernetes Repository

cat <<EOF > /etc/yum.repos.d/kubernetes.repo

[kubernetes]

name=Kubernetes

baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86\_64

enabled=1

gpgcheck=1

repo\_gpgcheck=1

gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg

exclude=kube\*

EOF

Installing Docker | Kubelet | Kubeadm | Kubectl

# yum update -y

# yum install -y docker kubeadm kubelet kubectl --disableexcludes=kubernetes

Start and enable docker and kubectl

# systemctl enable docker && systemctl start docker

# systemctl enable kubelet && systemctl start kubelet

For CentOS

# cat <<EOF > /etc/sysctl.d/k8s.conf

net.bridge.bridge-nf-call-ip6tables = 1

net.bridge.bridge-nf-call-iptables = 1

EOF

For RHEL

# sysctl net.bridge.bridge-nf-call-iptables=1

# sysctl net.ipv4.ip\_forward=1

# sysctl --system

# echo "1" > /proc/sys/net/ipv4/ip\_forward

Restart the systemd daemon and the kubelet service with the commands:

# systemctl daemon-reload

# systemctl restart kubelet

### 4. Configure Kubernetes "master" node

Initializing master node

# kubeadm init --pod-network-cidr=10.240.0.0/16

If you want to run kubectl as "regular" user. Then, execute below.

# mkdir -p $HOME/.kube

# sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

# sudo chown $(id -u):$(id -g) $HOME/.kube/config

Note down Join command in below step.

Installing Flannel network-plug-in for cluster network

<https://gist.github.com/rkaramandi/44c7cea91501e735ea99e356e9ae7883>

**Note: Refer the above link for flannel setup rather executing the following command line.**

# kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/a70459be0084506e4ec919aa1c114638878db11b/Documentation/kube-flannel.yml

# kubectl get pods --all-namespaces

### 5. Join "worker" nodes to the cluster

Get exact join command from previous kubeadm init command output.

# kubeadm join 192.168.33.9:6443 --token 967hcw.0ldqs3mjqurz10sx --discovery-token-ca-cert-hash sha256:9a4d1e595e5a595f9ceb9249030a605639a59997d1899704a09cd06878f2ee06

### 6. Testing

Display nodes status

# kubectl get no

Deploying sample application

# kubectl apply -f https://raw.githubusercontent.com/kubernetes/website/master/content/en/examples/controllers/nginx-deployment.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deployment

labels:

app: nginx

spec:

replicas: 3

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx:1.14.2

ports:

- containerPort: 80

Displaying Pod status

# kubectl get po -o wide

# **PODS**

### 1. https://labs.play-with-k8s.com/

# nginx-pod.yaml

apiVersion: v1

kind: Pod

metadata:

name: nginx-pod

labels:

app: nginx

tier: dev

spec:

containers:

- name: nginx-container

image: nginx

### 2. Create and display Pods

Create and display PODs

# kubectl create -f nginx-pod.yaml

# kubectl get pod

# kubectl get pod -o wide

# kubectl get pod nginx-pod -o yaml

# kubectl describe pod nginx-pod

### 3. Test & Delete

To get inside the pod

# kubectl exec -it nginx-pod -- /bin/sh

Create test HTML page

# cat <<EOF > /usr/share/nginx/html/test.html

<!DOCTYPE html>

<html>

<head>

<title>Testing..</title>

</head>

<body>

<h1 style="color:rgb(90,70,250);">Hello, Kubernetes...!</h1>

<h2>Congratulations, you passed :-) </h2>

</body>

</html>

EOF

exit

Expose PODS using NodePort service

kubectl expose pod nginx-pod --type=NodePort --port=80

Display Service and find NodePort

kubectl describe svc nginx-pod

Open Web-browser and access webapge using

<http://nodeip:nodeport/test.html>

Delete pod & svc

kubectl delete svc nginx-pod

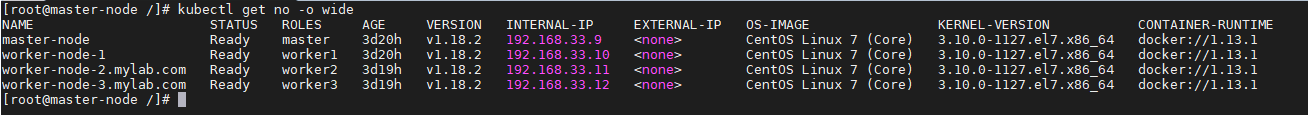
kubectl delete pod nginx-pod

Add Label

kubectl label node <node name> node-role.kubernetes.io/<role name>=<key - (any name)>

Remove label

kubectl label node <node name> node-role.kubernetes.io/<role name>-



# **Configuration**

## **ConfigMaps:**

### Overview:

1. Creating Configmap from "multiple files" & Consuming it inside Pod from "volumes"

1a. Create Configmap "nginx-configmap-vol" from "multiple files"

1b. Consume "nginx-configmap-vol" configmap inside Pod from "volumes"

1c. Create | Display | Validate

2. Creating Configmap from "literal values" & Consuming it inside Pod from "environment variables"

2a. Create Configmap “redis-configmap-env” from "literal values"

2b. Consume “redis-configmap-env” Configmap inside pod from “Environment Variables” inside pod

2c. Create | Display | Validate

3. Cleanup

3a. Delete configmaps

3b. Delete pods

3c. Validate

### 1. Creating Configmap from "multiple files" & Consuming it inside Pod from "volumes"

#### 1a. Create Configmap "nginx-configmap-vol" from "multiple files":

echo -n 'Non-sensitive data inside file-1' > file-1.txt

echo -n 'Non-sensitive data inside file-2' > file-2.txt

kubectl create configmap nginx-configmap-vol --from-file=file-1.txt --from-file=file-2.txt

# rm -f file-1 file-2

kubectl get configmaps

kubectl describe configmaps nginx-configmap-vol

#### 1b. Consume above "nginx-configmap-vol" configmap inside Pod from "volumes"

#nginx-pod-configmap-vol.yaml

apiVersion: v1

kind: Pod

metadata:

name: nginx-pod-configmap-vol

spec:

containers:

- name: nginx-container

image: nginx

volumeMounts:

- name: test-vol

mountPath: "/etc/non-sensitive-data"

readOnly: true

volumes:

- name: test-vol

configMap:

name: nginx-configmap-vol

items:

- key: file-1.txt

path: file-a.txt

- key: file-2.txt

path: file-b.txt

#### 1c. Create | Display | Validate:

# Create

kubectl create -f nginx-pod-configmap-vol.yaml

# Display

kubectl get po

kubectl get configmaps

kubectl describe pod nginx-pod-configmap-vol

# Validate from "inside" the pod

kubectl exec nginx-pod-configmap-vol -it /bin/sh

cd /etc/non-sensitive-data

ls

cat Non-sensitive data inside file-1.txt

cat password.txt

exit

(OR)

# Validate from "outside" the pod

kubectl exec nginx-pod-configmap-vol ls /etc/non-sensitive-data

kubectl exec nginx-pod-configmap-vol cat /etc/non-sensitive-data/file-a.txt

kubectl exec nginx-pod-configmap-vol cat /etc/non-sensitive-data/file-b.txt

### 2. Creating Configmap from "literal values" & Consuming it inside Pod from "environment variables"

#### 2a. Create configmap “redis-configmap-env” from "literal values"

kubectl create configmap redis-configmap-env --from-literal=file.1=file.a --from-literal=file.2=file.b

kubectl get configmap

kubectl describe configmap redis-configmap-env

#### 2b. Consume “redis-configmap-env” configmap inside pod from “Environment Variables” inside pod

# redis-pod-configmap-env.yaml

apiVersion: v1

kind: Pod

metadata:

name: redis-pod-configmap-env

spec:

containers:

- name: redis-container

image: redis

env:

- name: FILE\_1

valueFrom:

configMapKeyRef:

name: redis-configmap-env

key: file.1

- name: FILE\_2

valueFrom:

configMapKeyRef:

name: redis-configmap-env

key: file.2

restartPolicy: Never

#### 2c. Create | Display | Validate:

# Create

kubectl create -f redis-pod-configmap-env.yaml

# Display

kubectl get pods

kubectl get configmaps

kubectl describe pod redis-pod-configmap-env

# Validate from "inside" the pod

kubectl exec redis-pod-configmap-env -it /bin/sh

env | grep FILE

exit

(OR)

# Validate from "outside" the pod

kubectl exec redis-pod-configmap-env env | grep FILE

### 3. Cleanup

# Delete configmaps

kubectl delete configmaps nginx-configmap-vol redis-configmap-env

# Delete pods

kubectl delete pods nginx-pod-configmap-vol redis-pod-configmap-env

# Validate

kubectl get pods

kubectl get configmaps

## **Secrets**

### Overview:

1. Create Secret using "kubectl" & Consuming it from "volumes" inside Pod

1a. Create secret "nginx-secret-vol" using "Kubectl"

1b. Consume "nginx-secret-vol" from "volumes" inside Pod

1c. Create | Display | Validate

2. Create Secret "manually" using YAML file & Consuming it from "environment variables" inside Pod

2a. Create secret “redis-secret-env” using YAML file:

2b. Consume “redis-secret-env” secret from “Environment Variables” inside pod

2c. Create | Display | Validate

3. Cleanup

3a. Delete secrets

3b. Delete pods

3c. Validate

### 1. Creating Secret using Kubectl & Consuming it from "volumes" inside Pod

#### 1a. Creating secret using "Kubectl":

echo -n 'admin' > username.txt

echo -n 'pa$$w00rd' > password.txt

kubectl create secret generic nginx-secret-vol --from-file=username.txt --from-file=password.txt

# rm -f username.txt password.txt

kubectl get secrets

kubectl describe secrets nginx-secret-vol

#### 1b. Consuming "nginx-secret-vol" from "volumes" inside Pod

#nginx-pod-secret-vol.yaml

apiVersion: v1

kind: Pod

metadata:

name: nginx-pod-secret-vol

spec:

containers:

- name: nginx-container

image: nginx

volumeMounts:

- name: test-vol

mountPath: "/etc/confidential"

readOnly: true

volumes:

- name: test-vol

secret:

secretName: nginx-secret-vol

#### 1c. Create | Display | Validate:

# Create

kubectl create -f nginx-pod-secret-vol.yaml

# Display

kubectl get po

kubectl get secrets

kubectl describe pod nginx-pod-secret-vol

# Validate from "inside" the pod

kubectl exec nginx-pod-secret-vol -it /bin/sh

cd /etc/confidential

ls

cat username.txt

cat password.txt

exit

(OR)

# Validate from "outside" the pod

kubectl exec nginx-pod-secret-vol ls /etc/confidential

kubectl exec nginx-pod-secret-vol cat /etc/confidential/username.txt

kubectl exec nginx-pod-secret-vol cat /etc/confidential/password.txt

### 2. Creating Secret "manually" using YAML file & Consuming it from "environment variables" inside Pod

#### 2a. Creating Secret using YAML file:

# Encoding secret

echo -n 'admin' | base64

echo -n 'pa$$w00rd' | base64

# YAML file

# redis-secret-env.yaml

apiVersion: v1

kind: Secret

metadata:

name: redis-secret-env

type: Opaque

data:

username: YWRtaW4=

password: cGEkJHcwMHJk

kubectl create -f redis-secret-env.yaml

kubectl get secret

kubectl describe secret redis-secret-env

#### 2b. Consuming “redis-secret-env” secret from “Environment Variables” inside pod

# redis-pod-secret-env.yaml

apiVersion: v1

kind: Pod

metadata:

name: redis-pod-secret-env

spec:

containers:

- name: redis-container

image: redis

env:

- name: SECRET\_USERNAME

valueFrom:

secretKeyRef:

name: redis-secret-env

key: username

- name: SECRET\_PASSWORD

valueFrom:

secretKeyRef:

name: redis-secret-env

key: password

restartPolicy: Never

#### 2c. Create | Display | Validate:

# Create

kubectl create -f redis-pod-secret-env.yaml

# Display

kubectl get pods

kubectl get secrets

kubectl describe pod redis-pod-secret-env

# Validate from "inside" the pod

kubectl exec redis-pod-secret-env -it /bin/sh

env | grep SECRET

exit

(OR)

# Validate from "outside" the pod

kubectl exec redis-pod-secret-env env | grep SECRET

### 3. Cleanup

# Delete secrets

kubectl delete secrets nginx-secret-vol redis-secret-env

# Delete pods

kubectl delete pods nginx-pod-secret-vol redis-pod-secret-env

# Validate

kubectl get pods

kubectl get secrets

# **Controllers**

## **Replication Controller**

### 1. Replication Controller YAML file

# nginx-rc.yaml

apiVersion: v1

kind: ReplicationController

metadata:

name: nginx-rc

spec:

replicas: 3

template:

metadata:

name: nginx-pod

labels:

app: nginx-app

spec:

containers:

- name: nginx-container

image: nginx

ports:

- containerPort: 80

selector:

app: nginx-app

### 2. Create and display

kubectl create -f nginx-rc.yaml

kubectl get po -o wide

kubectl get po -l app=nginx-app

kubectl get rc nginx-rc

kubectl describe rc nginx-rc

### 3. Reschedule

kubectl get po -o wide --watch

kubectl get po -o wide

kubectl get nodes

### 4. Scaling up cluster

kubectl scale rc nginx-rc --replicas=5

kubectl get rc nginx-rc

kubectl get po -o wide

### 5. Scaling down

kubectl scale rc nginx-rc --replicas=3

kubectl get rc nginx-rc

kubectl get po -o wide

### 6. Cleanup

kubectl delete -f nginx-rc.yaml

kubectl get rc

kubectl get po -l app=nginx-app

## **Replica Set**

### ReplicaSet YAML file

# nginx-rs.yaml

apiVersion: apps/v1

kind: ReplicaSet

metadata:

name: nginx-rs

spec:

replicas: 3

template:

metadata:

name: nginx-pod

labels:

app: nginx-app

tier: frontend

spec:

containers:

- name: nginx-container

image: nginx

ports:

- containerPort: 80

selector:

matchLabels:

app: nginx-app

matchExpressions:

- {key: tier, operator: In, values: [frontend]}

### Create and display replicaset

kubectl create -f nginx-rs.yaml

kubectl get po -o wide

kubectl get po -l app=nginx-app

kubectl get rs nginx-rs -o wide

kubectl describe rs nginx-rs

### Automatic Pod Reschedule

kubectl get po -o wide --watch

kubectl get po -o wide

kubectl get nodes

### Scale up pods

kubectl scale rs nginx-rs --replicas=5

kubectl get rs nginx-rs -o wide

kubectl get po -o wide

### Scale down pods

kubectl scale rs nginx-rs --replicas=3

kubectl get rs nginx-rs -o wide

kubectl get po -o wide

### Cleanup

kubectl delete -f nginx-rs.yaml

kubectl get rs

kubectl get po -l app=nginx-app

## **Deployment**

### Deployment YAML file

# nginx-deploy.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deploy

labels:

app: nginx-app

spec:

replicas: 3

template:

metadata:

labels:

app: nginx-app

spec:

containers:

- name: nginx-container

image: nginx:1.7.9

ports:

- containerPort: 80

selector:

matchLabels:

app: nginx-app

### Create and Display Deployment

kubectl create -f nginx-deploy.yaml

kubectl get deploy -l app=nginx-app

kubectl get rs -l app=nginx-app

kubectl get po -l app=nginx-app

kubectl describe deploy nginx-deploy

### Testing: Rollback update

kubectl set image deploy nginx-deploy nginx-container=nginx:1.91 --record

kubectl rollout status deployment/nginx-deploy

kubectl rollout history deployment/nginx-deploy

kubectl rollout undo deployment/nginx-deploy

kubectl rollout status deployment/nginx-deploy

kubectl describe deploy nginx-deploy | grep -i image

### Testing: Update Version of "nginx:1.7.9" to "nginx:1.9.1"

kubectl set image deploy nginx-deploy nginx-container=nginx:1.9.1

kubectl edit deploy nginx-deploy

kubectl rollout status deployment/nginx-deploy

kubectl get deploy

### Testing: Scale UP

kubectl scale deployment nginx-deploy --replicas=5

kubectl get deploy

kubectl get po -o wide

### Testing: Scale DOWN

kubectl scale deployment nginx-deploy --replicas=3

kubectl get deploy

kubectl get po -o wide

### Cleanup

kubectl delete -f nginx-deploy.yaml

kubectl get deploy

kubectl get rs

kubectl get po

## **DaemonSet**

### Overview:

1. Deploy Pod on "all" worker nodes inside k8s cluster using DaemonSet

1a. fluentd-DaemonSet manifest file

1b. Create | Display | Validate

2. Deploy Pod on "Subset" of worker nodes inside k8s cluster using DaemonSet

2a. Attach label to the nodes

2b. nginx-DamoneSet manifest file review

2c. Create | Display | Validate

3. Cleanup

### 1. Deploy Pod on "all" worker nodes inside k8s cluster using DaemonSet

#### 1a. YAML File:

# fluentd-ds-allnodes.yaml

apiVersion: apps/v1

kind: DaemonSet

metadata:

name: fluentd-ds

spec:

template:

metadata:

labels:

name: fluentd

spec:

containers:

- name: fluentd

image: gcr.io/google-containers/fluentd-elasticsearch:1.20

selector:

matchLabels:

name: fluentd

#### 1b. Create | Display | Validate

kubectl create -f fluentd-ds-allnodes.yaml

kubectl get po -o wide

kubectl get ds

kubectl describe ds fluentd-ds

### 2. Deploy Pod on "Subset" of worker nodes inside k8s cluster using DaemonSet

#### 2a. Attach label to the nodes

kubectl get nodes

kubectl label nodes worker1 worker2 disktype=ssd

kubectl get nodes --show-labels

#### 2b. YAML

# nginx-ds-subsetnodes.yaml

apiVersion: apps/v1

kind: DaemonSet

metadata:

name: nginx-ds

spec:

template:

metadata:

labels:

name: nginx

spec:

containers:

- name: nginx-container

image: nginx

nodeSelector:

disktype: ssd

selector:

matchLabels:

name: nginx

#### 2c. Create | Display | Validate

kubectl create -f nginx-daemonset.yaml

kubectl get po -o wide

kubectl get ds

kubectl describe ds nginx-ds

### 3. Cleanup

kubectl delete ds fluentd-ds

kubectl delete ds nginx-ds

kubectl get po

## **Job**

### 1. Job manifest file:

apiVersion: batch/v1

kind: Job

metadata:

name: countdown

spec:

template:

metadata:

name: countdown

spec:

containers:

- name: counter

image: centos:7

command:

- "bin/bash"

- "-c"

- "for i in 9 8 7 6 5 4 3 2 1 ; do echo $i ; done"

restartPolicy: Never

### 2. Create & Display

kubectl create –f countdown-jobs.yaml

kubectl get jobs

kubectl get po

kubectl describe jobs countdown

### 3. Test

kubectl logs [POD\_NAME]

### 4. Cleanup

kubectl delete jobs countdown

kubectl get po

# **Service**

## **NodePort Service**

### 1. Deployment & NodePort service manifest file

Deployment YAML file:

# Deployment

# nginx-deploy.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deployment

labels:

app: nginx-app

spec:

replicas: 1

selector:

matchLabels:

app: nginx-app

template:

metadata:

labels:

app: nginx-app

spec:

containers:

- name: nginx-container

image: nginx:1.7.9

ports:

- containerPort: 80

NodePort Service YAML file:

# Service

# nginx-svc-np.yaml

apiVersion: v1

kind: Service

metadata:

name: my-service

labels:

app: nginx-app

spec:

selector:

app: nginx-app

type: NodePort

ports:

- nodePort: 31111

port: 80

targetPort: 80

### 2. Create and Display Deployment and NodePort

kubectl create –f nginx-deploy.yaml

kubectl create -f nginx-svc.yaml

kubectl get service -l app=nginx-app

kubectl get po -o wide

kubectl describe svc my-service

### 3. Testing

# To get inside the pod

kubectl exec [POD-IP] -it /bin/sh

Create test HTML page

cat <<EOF > /usr/share/nginx/html/test.html

<!DOCTYPE html>

<html>

<head>

<title>Testing..</title>

</head>

<body>

<h1 style="color:rgb(90,70,250);">Hello, NodePort Service...!</h1>

<h2>Congratulations, you passed :-) </h2>

</body>

</html>

EOF

exit

Test using Pod IP:

kubectl get po -o wide

curl http://[POD-IP]/test.html

NodePort – Accessing using Service IP

Test using Service IP:

kubectl get svc -l app=nginx-app

curl http://[cluster-ip]/test.html

Test using Node IP (external IP)

http://nodep-ip:nodePort/test.html

note: node-ip is the external ip address of a node.

### 4. Cleanup

kubectl delete -f nginx-deploy.yaml

kubectl delete -f nginx-svc.yaml

kubectl get deploy

kubectl get svc

kubectl get pods

## **Load Balancer Service**

### 1. YAML: Deployment & Load Balancer Service

Deployment

# controllers/nginx-deploy.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deployment

labels:

app: nginx-app

spec:

replicas: 1

selector:

matchLabels:

app: nginx-app

template:

metadata:

labels:

app: nginx-app

spec:

containers:

- name: nginx-container

image: nginx:1.7.9

ports:

- containerPort: 80

Service - LoadBalancer

#lb.yaml

apiVersion: v1

kind: Service

metadata:

name: my-service

labels:

app: nginx-app

spec:

selector:

app: nginx-app

type: LoadBalancer

ports:

- nodePort: 31000

port: 80

targetPort: 80

### 2. Create & Display: Deployment & Load Balancer Service

kubectl create –f nginx-deploy.yaml

kubectl create -f lb.yaml

kubectl get pod -l app=nginx-app

kubectl get deploy -l app=nginx-app

kubectl get service -l app=nginx-app

kubectl describe service my-service

### 3. Testing Load Balancer Service

# To get inside the pod

kubectl exec -it [pod-name] -- /bin/sh

# Create test HTML page

cat <<EOF > /usr/share/nginx/html/test.html

<!DOCTYPE html>

<html>

<head>

<title>Testing..</title>

</head>

<body>

<h1 style="color:rgb(90,70,250);">Hello, Kubernetes...!</h1>

<h2>Load Balancer is working successfully. Congratulations, you passed :-) </h2>

</body>

</html>

EOF

exit

# Test using load-balancer-ip

http://load-balancer-ip

http://load-balancer-ip/test.html

# Testing using nodePort

http://nodeip:nodeport

http://nodeip:nodeport/test.html

### 4. Cleanup

kubectl delete –f nginx-deploy.yaml

kubectl delete -f lb.yaml

kubectl get pod

kubectl get deploy

kubectl get service

## **ClusterIP Service**

OVERVIEW:

Step-1: Set up a "Redis master"

1a. Create redis-master "deployment"

1b. Create redis-master "service"

Step-2: Set up a "Redis slave"

2a. Create redis-master "deployment"

2b. Create redis-master "slave"

Step 3: Set up the "guestbook web frontend"

3a. Create guestbook web frontend "deployment"

3c. Expose frontend on an external IP address (LoadBalancer)

### Step-1: Set up a "Redis master"

#### a. Create redis-master "deployment"

redis-master-deployment.yaml

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

name: redis-master

spec:

replicas: 1

template:

metadata:

labels:

app: redis

role: master

tier: backend

spec:

containers:

- name: master

image: k8s.gcr.io/redis:e2e # or just image: redis

resources:

requests:

cpu: 100m

memory: 100Mi

ports:

- containerPort: 6379

#### b. Create redis-master "service"

# redis-master-service.yaml

apiVersion: v1

kind: Service

metadata:

name: redis-master

labels:

app: redis

role: master

tier: backend

spec:

type: ClusterIP

ports:

- port: 6379

targetPort: 6379

selector:

app: redis

role: master

tier: backend

kubectl create -f redis-master-deployment.yaml

kubectl create -f redis-master-service.yaml

kubectl get deploy

kubectl get svc

kubectl get pods

kubectl get logs -f [pod-name]

### Step 2: Set up Redis "slaves"

#### 2a. Create redis-slave "deployment"

# redis-slave-deployment.yaml

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

name: redis-slave

spec:

replicas: 2

template:

metadata:

labels:

app: redis

role: slave

tier: backend

spec:

containers:

- name: slave

image: gcr.io/google\_samples/gb-redisslave:v1

resources:

requests:

cpu: 100m

memory: 100Mi

env:

- name: GET\_HOSTS\_FROM

value: dns

# If your cluster config does not include a dns service, then to

# instead access an environment variable to find the master

# service's host, comment out the 'value: dns' line above, and

# uncomment the line below:

# value: env

ports:

- containerPort: 6379

#### 2b. Create redis-slave "service"

# redis-slave-service.yaml

apiVersion: v1

kind: Service

metadata:

name: redis-slave

labels:

app: redis

role: slave

tier: backend

spec:

ports:

- port: 6379

selector:

app: redis

role: slave

tier: backend

kubectl create -f redis-slave-deployment.yaml

kubectl create -f redis-slave-service.yaml

kubectl get deploy

kubectl get svc

kubectl get pods

### Step 3: Set up the guestbook web frontend

#### 3a. Create fronend "deployment"

# frontend-deployment.yaml

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

name: frontend

spec:

replicas: 2

template:

metadata:

labels:

app: guestbook

tier: frontend

spec:

containers:

- name: php-redis

image: gcr.io/google-samples/gb-frontend:v4

resources:

requests:

cpu: 100m

memory: 100Mi

env:

- name: GET\_HOSTS\_FROM

value: dns

# If your cluster config does not include a dns service, then to

# instead access environment variables to find service host

# info, comment out the 'value: dns' line above, and uncomment the

# line below:

# value: env

ports:

- containerPort: 80

3b. Expose frontend on an external IP address

# frontend-service.yaml

apiVersion: v1

kind: Service

metadata:

name: frontend

labels:

app: guestbook

tier: frontend

spec:

# if your cluster supports it, uncomment the following to automatically create

# an external load-balanced IP for the frontend service.

type: LoadBalancer

ports:

- port: 80

selector:

app: guestbook

tier: frontend

Create & display "frontend" deployment and service

kubectl create -f frontend-deployment.yaml

kubectl create -f frontend-service.yaml

kubectl get deploy

kubectl get svc

kubectl get pods

kubectl get service frontend

[Web-browser] - http://[LB-IP]

### 4. Cleanup:

# Delete redis-master

kubectl delete -f redis-master-deployment.yaml

kubectl delete -f redis-master-service.yaml

# Delete redis-slave

kubectl delete -f redis-slave-deployment.yaml

kubectl delete -f redis-slave-service.yaml

# Delete frontend-app

kubectl delete -f frontend-deployment.yaml

kubectl delete -f frontend-service.yaml

# Display

kubectl get deploy

kubectl get svc

kubectl get pods

# **Storage**

## **emptyDir**

### 1. Pod with emptyDir Volume YAML (example)

# nginx-emptydir.yaml

apiVersion: v1

kind: Pod

metadata:

name: nginx-emptydir

spec:

containers:

- name: nginx-container

image: nginx

volumeMounts:

- name: test-vol

mountPath: /test-mnt

volumes:

- name: test-vol

emptyDir: {}

### 2. Create & Display Pod with emptyDir volume

kubectl create -f nginx-emptydir.yaml

kubectl get po -o wide

kubectl exec nginx-emptydir df /test-mnt

kubectl describe pod nginx-emptydir

### 3. Cleanup

kubectl delete po nginx-emptydir

## **HostPath**

### 1. HostPath YAML file

apiVersion: v1

kind: Pod

metadata:

name: nginx-hostpath

spec:

containers:

- name: nginx-container

image: nginx

volumeMounts:

- mountPath: /test-mnt

name: test-vol

volumes:

- name: test-vol

hostPath:

path: /test-vol

### 2. Create and Display HostPath

kubectl create -f nginx-hostpath.yaml

kubectl get po

kubectl exec nginx-hostpath df /test-mnt

### 3. Test: Creating "test" file underlying host dir & accessing from from pod

From HOST:

cd /test-vol

echo "From Host" > from-host.txt

cat from-host.txt

From POD:

kubectl exec nginx-hostpath cat /test-mnt/from-host.txt

### 4. Test: Creating "test" file inside the POD & accessing from underlying host dir

From POD:

kubectl exec nginx-hostpath -it -- /bin/sh

cd /test-mnt

echo "From Pod" > from-pod.txt

cat from-pod.txt

From Host:

cd /test-vol

ls

cat from-pod.txt

### 5. Clean up

kubectl delete po nginx-hostpath

kubectl get po

ls /test-vol

## **gce Persistent Disk**

### 1. Create Disk on Google Cloud

From Dashboard

(OR)

gcloud compute disks create --size=10GB --zone=us-central1-a my-data-disk

gcloud compute disks list

note: gcloud auth login

### 2. Pod with gcePersistentDisk YAML file

apiVersion: v1

kind: Pod

metadata:

name: gce-pd

spec:

containers:

- name: nginx-container

image: nginx

volumeMounts:

- mountPath: /test-pd

name: test-volume

volumes:

- name: test-volume

gcePersistentDisk:

pdName: my-data-disk

fsType: ext4

*Bug Fix: For deploying Pod with gcePersistentDisk on GCE "VMs"*

*a. Manually attach above created disk to worker node through GCE dashboard*

*b. Format above disk on the respective worker node where the above pod deployed*

*mkfs.ext4 /dev/disk/by-id/scsi-0Google\_PersistentDisk\_{Persistent Disk Name}*

*c Mount it in the location expected by Kubernetes. Run the following commands as root:*

*mkdir -p /var/lib/kubelet/plugins/kubernetes.io/gce-pd/mounts/{Persistent Disk Name} && mount /dev/disk/by-id/scsi-0Google\_PersistentDisk\_{Persistent Disk Name} /var/lib/kubelet/plugins/kubernetes.io/gce-pd/mounts/{Persistent Disk Name}*

*Reference: https://access.redhat.com/documentation/en-us/red\_hat\_enterprise\_linux\_atomic\_host/7/html/getting\_started\_with\_kubernetes/get\_started\_provisioning\_storage\_in\_kubernetes*

### 3. Create and Display

kubectl create -f gcepd.yaml

kubectl get po –o wide

kubectl describe po gce-pd

### 4. Testing: Creating file inside mount where gcePersistentDisk volume is mounted and then delete

Create test file inside pod

kubectl exec gce-pd -it -- /bin/sh

df

echo "Testing - 1" > /test-pd/test1.html

exit

Delete pod:

kubectl delete -f gce-pd.yaml

### 5. Validate: Recreate Pod using same gcePersistentDisk above and validate above data

Recreate pod with same config

kubectl create -f gce-pd.yaml

kubectl get po –o wide

Validate test file created in step#4 is still available?

kubectl exec gce-pd -it /bin/sh

ls /test-pd/

cat /test-pd/test1.html

### 6. Cleanup

kubectl delete -f gce-pd.yaml

kubectl get pods

## **Static Persistent Volumes**

### 1. Create gcePersistent Disk on Google Cloud

From Dashboard

(OR)

gcloud compute disks create --size=10GB --zone=us-central1-a my-data-disk

gcloud compute disks list

note: create gcePersistentdisk in same zone as k8 cluster is in

### 2. Create "Persistent Volume(PV)" and display

YAML

apiVersion: v1

kind: PersistentVolume

metadata:

name: pv-gce

spec:

capacity:

storage: 15Gi

accessModes:

- ReadWriteOnce

storageClassName: slow

gcePersistentDisk:

pdName: my-data-disk

fsType: ext4

Create:

kubectl create -f pv.yaml

kubectl get pv

kubectl describe pv pv-gce

### 3. Create "Persistent Volume Claim(PVC)" and display

YAML:

# pvc.yaml

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: my-disk-claim

spec:

resources:

requests:

storage: 15Gi

accessModes:

- ReadWriteOnce

storageClassName: slow

Create & Display:

kubectl create -f pvc.yaml

kubectl get pvc

kubectl get pv

### 4. Reference claim inside the Pod:

# nginx-pv.yaml

apiVersion: v1

kind: Pod

metadata:

name: pv-pod

spec:

containers:

- name: test-container

image: nginx

volumeMounts:

- mountPath: /test-pd

name: test-volume

volumes:

- name: test-volume

persistentVolumeClaim:

claimName: my-disk-claim

Create & Display:

kubectl create -f nginx-pv.yaml

kubectl get po -o wide

### 5. Testing

#### a. Create a test file inside the mount where the disk is mounted

kubectl exec pv-pod -it -- /bin/sh

df /test-pd

cd /test-pd

echo "From first pod" > test1.txt

exit

#### b. Delete the Pod

kubectl delete -f nginx-pv.yaml

#### c. Recreate the Pod with same configuration

kubectl create -f nginx-pv.yaml

kubectl get pods -o wide

#### d. Verify the data created in step-1 is still available

kubectl exec pv-pod df /test-pd

kubectl exec pv-pod ls /test-pd/

kubectl exec pv-pod cat /test-pd/test1.html

### 6. Cleanup

kubectl delete -f nginx-pv.yaml

kubectl delete -f pvc.yaml

kubectl delete -f pv.yaml

kubectl get pods

kubectl get pvc

kubectl get pv

## **Dynamic Volume Provisioning**

### 1. Create Storage Classes

YAML:

# sc.yaml

kind: StorageClass

apiVersion: storage.k8s.io/v1

metadata:

name: fast

provisioner: kubernetes.io/gce-pd

parameters:

type: pd-ssd

Create & Display:

kubectl get sc

kubectl create -f sc.yaml

kubectl get sc

kubectl describe storageclass fast

### 2. Create Persistent Volume Claim (PVC)

# pvc1.yaml

kind: PersistentVolumeClaim

apiVersion: v1

metadata:

name: my-disk-claim-1

spec:

resources:

requests:

storage: 30Gi

accessModes:

- ReadWriteOnce

storageClassName: fast

# pvc2.yaml

kind: PersistentVolumeClaim

apiVersion: v1

metadata:

name: my-disk-claim-2

spec:

resources:

requests:

storage: 40Gi

accessModes:

- ReadWriteOnce

storageClassName: fast

Create & Display:

kubectl create -f pvc1.yaml

kubectl create -f pvc2.yaml

kubectl get pvc

### 3. Reference Claim inside Pod

# nginx-pv.yaml

apiVersion: v1

kind: Pod

metadata:

name: pv-pod

spec:

containers:

- name: test-container

image: nginx

volumeMounts:

- mountPath: /test-pd

name: test-volume

volumes:

- name: test-volume

persistentVolumeClaim:

claimName: my-disk-claim-1

Create & Display:

kubectl create -f nginx-pv.yaml

kubectl get po -o wide

### 4. Testing

#### a. Create a sample test file inside the mount.

kubectl exec pv-pod -it -- /bin/sh

df /test-pd

cd /test-pd

echo "From test-1" > test1.txt

exit

#### b. Delete the Pod

kubectl delete -f nginx-pv.yaml

kubectl get po –o wide

#### c. Recreate the Pod with same configuration

kubectl create -f nginx-pv.yaml

kubectl get po –o wide

#### d. Verify the data created in step-1 is still available?

kubectl exec pv-pod df /test-pd

kubectl exec pv-pod ls /test-pd/

kubectl exec pv-pod cat /test-pd/test1.txt

### 5.Cleanup

kubectl delete -f nginx-pv.yaml

kubectl delete -f pvc1.yaml

kubectl delete -f pvc2.yaml

kubectl delete -f sc.yaml

kubectl get pods

kubectl get pvc

kubectl get sc